The INTREPID SDK and API (R18)

Top

You can write your own software which accesses INTREPID datasets using the prototypes in the INTREPID API library. The API works for all INTREPID dataset types (grid, line, point, polygon).

The INTREPID API library is a low cost way for your in-house developers to have easy access to INTREPID datasets.

The INTREPID Software Development Kit is also available free of charge with INTREPID.

INTREPID Software Development Kit

The INTREPID Software Development Kit (SDK) is a simplified Application Programmer's Interface to the INTREPID binary database. It includes such operations as Open, Read/Write AddColumn, Multi-band data access.

The SDK consists of

- 15 short C source code example programs
- A suggested way to create executables that utilise the functions contained within the API.

You can use most compilers provided they produce object code that is compatable with the supplied dynamic linked library file (DLL for Windows or SO for Linux). This file is distributed on the CD-ROM for all platforms and is available free. The first **`test1`** code contains more detailed instructions and suggestions for getting started.

Full source code is available on request for the simplified SDK.

WorldWind

Intrepid has contributed extensions to WorldWind (NASA/Geoscience Australia). This treats the sample data and tutorials with any distribution of Intrepid, Jetstream and Geomodeller as explorable project datasets. A GIT repository is established for you to source the code from, and this is designed to work out of the box in Eclipse. The Eclipse framework is a very powerful development environment for creative people to tailor make new integrated tailored solutions.

In principle, the main Australian GADDS, Irish GSI, Namibian GS whole of country geophysical data catalogs can be accessed via this interface.

GOOGLE Protobuf

With V5.0 INTREPID and Geomodeller V2012, all the formal processes & tools data models are published with the API. This open ups all the designs and formal specifications. The "*.proto" files that are shipped are exactly the same ones used to power the batch interface to all the tools. The toolkit supplied by GOOGLE to create Java, Python and C++ wrapper libraries are used by INTREPID to access the task files. Sample task files in this format are distributed for every tool and function. Many previously undocumneted features become avialble, as this new policy guarentees what you see here is what is actually being used. Extensive commmentaries are also embedded with this new release. Processes, via this interface, work seemlessly across architectures, and clusters

Database structure notes

The basic database structure is designed to be fast, flexible and simple and uses a forward Index to point to start of each new group of data in each channel. There is always a 512 byte header before the data. This contains metadata that is now mostly duplicated in the secondary **.vec**, **.ers**, **.isi** files. This binary header came from the ERMapper/PITS heritage. However, we dropped the 512 byte header for grids quite some time ago.

Availability of the API library

The API library (called **libdfa.a** or **libdfa.so** under UNIX and **dfa.lib** under *Windows*) is distributed with INTREPID and normally locked. You can arrange to have this library unlocked for a small charge. We negotiate unlocking the library on a case by case basis.

Please contact us if you wish to use the API.

Also available is a test suite of C programs that demonstrates each of the functions. The test programs are named **test1**, ... and reside in the ***install_path*/usercode/io** directory (where ***install_path*** is the location of your INTREPID installation).

Methods of access to your code and the API

You can create programs of three types

- **Stand alone user programs** in the form of '**.exe**' files which can directly access INTREPID datasets using the API libraries.
- **User defined API functions**. These functions are compiled and linked code which
  - Have access to the API library.
  - Are called from the Spreadsheet, Subsection and other tools using their file names prefixed by an @ symbol,
  - Accept only the names of fields or grid bands as arguments,
  - Can directly access INTREPID datasets using the API libraries and
  - Return a temporary whole dataset field or grid band as the value of the function.
- **User defined DLL/SO functions**. These functions are compiled and linked code which
  - Reside in separate **.dll** or **.so** files each with the same name as the function it contains,
  - Are called from the Spreadsheet, Subsection and other tools using their file name without an @ symbol prefix,
  - Accept and process arguments representing the values of an expression for all data points in a dataset group and
  - Return values for all data points in the group as the value of the function.
- **External hard copy composition macros** These are programs which
  - You can execute using an external macro specification from within a MAPCOMP hard copy specification file.
  - Generate MAPCOMP language statements which INTREPID inserts in place of the external macro specification.

See "External macros" in MAPCOMP Map Specification Language (R20) for full details.

The two user defined function types use a dynamic link to the INTREPID function library (known as the INTREPID 'calc class'). Allowing for a minor restriction on user defined API function arguments, you can fully integrate user defined functions into INTREPID expression syntax. See INTREPID expressions and functions (R12) for a full description of this library.

Prototypes, data types and the API library

The files **dfatypes.h** and **dfa.h** contain full definitions of all data types and listings of prototypes required for the API. They are available in the directory ***install_path*/sample_data/examples/usercode/io** (where ***install_path*** is the location of your INTREPID installation). You must include **dfa.h** in all programs that have API access.

The API library is available as follows.

Under *Windows* the API is in a static library called **dfa.lib** residing in the ***install_path*/bin/win32** directory.

Under UNIX both static and shared object versions are available. The static version is called **libdfa.a**. The shared object version is called **libdfa.so**. The libraries reside in the ***install_path*/bin/platform/lib** directory. (Under Solaris, for example, this directory is ***install_path*/bin/SOLARIS/lib**. See your installation notes and Section "Configuring INTREPID under Solaris and SunOS" in Configuring and using INTREPID (R04) for further details.)

Stand alone user programs

>> To create a stand alone user program with API access

1. Create a program which uses the API data types and prototypes correctly and which has an **#include "dfa.h"** statement. The program may have command line parameters as required.
2. Compile the program.
3. Link the program using the API library and produce a '**.exe**' file (the extension **.exe** is not strictly required under UNIX).

>> To execute a stand alone user program with API access

1. *(Under UNIX)* Ensure that **libdfaioapi.so** resides in a directory specified in the system parameter LD_LIBRARY_PATH (e.g., ***install_path*/bin/SOLARIS/lib**, where ***install_path*** is the location of your INTREPID installation.

   (Under Windows) Ensure that **dfaioapi.lib** resides in the directory ***install_path*/bin/win32** (where ***install_path*** is the location of your INTREPID installation), or at least is available in a directory listed in the PATH.
2. Use the program name as a command, specifying command line parameters as required by the program.

Example 1

This program has one command line parameter representing a vector dataset.

```
surv_update.exe  /disk1/surv/ebag_A
```

Example 2

You can find an example of a stand alone user defined program (**test1.c**) with API access in the directory ***install_path*/examples/usercode/io/test1**

User defined API functions

>> To create a user defined API function

1 Create a program which uses the API data types and prototypes correctly and which has an **`#include "dfa.h"`** statement. The first command line parameter corresponds to the output data. The second and subsequent 'command line' parameters correspond to the arguments that you will use in the function call.

The function call

`@function_name(input1, input2, input3 ...)`

Would result in INTREPID executing your program with the command line

`function_name output input1 input2 input3 ...`

Where *`output`* becomes the value passed back to INTREPID and returned as the value of the function.

Note: You must prefix the function name with an **`@`** symbol when calling the function from an INTREPID tool.

2 Compile the program.

3 Link the program using the API library and produce a file with exactly the same name as the function you will be calling.

>> To call a user defined API function

Call the function from an INTREPID tool using the format described above.

You may only use field or grid band names as arguments when calling API functions. INTREPID does not accept expressions as arguments for user defined API functions.

Example 1

You have a function call in an INTREPID tool:

```
@StdVal(Mag1, Mag2)
```

This corresponds to your user defined program

```
StdVal
```

INTREPID executes the program with command line parameters as follows

`StdVal` **`output_data`** Mag1 Mag2

When the program has finished, INTREPID obtains the data from *`output_data`* and returns it as the value of the function in the tool.

Example 2

You can find an example of a user defined API function (**`test1.c`**) in the directory ***`install_path`*`/examples/usercode/dbedit/test1`**, including a sample Spreadsheet tool task specification (**`.job`**) file which shows how to call the function.

```
Action Begin
        Type    = CreateField
        Name    = aMAGNETIC
        Dtype   = IEEE8ByteReal
        Initial = "@test1(MAGNETIC)"
Action End
```

User defined DLL/SO functions

>> To create a user defined DLL/SO function

1 Create a program defining a function which processes the arguments according to your requirements. The first argument in the program function definition specifies the number of data points in the group being processed. The second and subsequent arguments in the program function correspond to the arguments that you will use in the function call.

The function call

`function_name(input1, input2, ...)`

would require a function in your program result in INTREPID executing the following function in your program

`function_name(no_of_points, input1, input2, ...)`

Where *`no_of_points`* is the number of data points in the group represented by the arguments.

Note: You must NOT prefix the function name with an `@` symbol when calling the function from an INTREPID tool.

2 Compile the program.

3 Link the program and produce a file with the same name as the function you will be calling and a linked library / shared object extension.

Under Windows a function called **`CompVal`** would reside in a file **`CompVal.dll`**. This file must be located in a directory specified in the environment variable PATH.

Note: You must 'publish' the **`Double*`** function prototype. For Example 1 below, you would include the following statement in the header specifications

`external publish double* CompVal(int no_pts,double* ZVal1,double* ZVal2)`*Under UNIX* a function called **`CompVal`** would reside in a file **`CompVal.so`**. This file must be located in a directory specified by the system parameter LD_LIBRARY_PATH.

>> To call a user defined DLL/SO function

Call the function from an INTREPID tool using the format described in the steps above.

You may use field or grid band name or expressions containing field or grid band names as arguments.

___

Example 1

You have a function call in an INTREPID tool:

```
CompVal(Mag1, Mag2)
```

INTREPID will access your user defined program called

`CompVal.dll` or **`CompVal.so`**.

In the program is the following function

```
double* CompVal(int no_pts,double* ZVal1,double* ZVal2)
```

Example 2

You can find an example of a user defined DLL/SO function (**test2.c**) in ***install_path*/examples/usercode/dbedit/test2** (where ***install_path*** is the location of your INTREPID installation), including a sample Spreadsheet tool task specification (**.job**) file which shows how to call the function.

```
Action Begin
        Type    = CreateField
        Name    = aMAGNETIC
        Dtype   = IEEE8ByteReal
        Initial = "test2(MAGNETIC)"
Action End
```

Multiple function DLL/SO library files

You can have several user-defined functions in a single **.dll** or **.so** library file. Instead of using a matching file name to locate the name of the function, you must specify each library file in **Library Begin–End** blocks in **install.cfg**. in the **config** directory (for example, **d:\intrepid\config**).

A library definition block contains

- A **Name =** statement which specifies the library file name.
- A **Module Begin–End** block for each function.

A **Module Begin–End** block contains

- A **Name =** statement corresponding to the function name.
- **Input =** statements describing each argument of the function. A name you specify for an argument here is purely for documentation. INTREPID does not use the name.
- An **Output =** statement describing the output of the function. A name you specify for the output here is purely for documentation. INTREPID does not use the name.

Here is an example, the **gravity** library:

```
Library Begin
   Name = gravity
   Module Begin
      Name = FA
      Input = Latitude
      Input = ObsGravity
      Output = FreeAir
   Module End
   Module Begin
      Name = FAmgal
      Input = Latitude
      Input = "ObsGravity(m/sec^2)"
      Output = FreeAir
   Module End
Library End
```

- The library resides in the file **gravity.dll** or **gravity.so**.
- It contains two functions.

FA(Latitude,ObsGravity), with two arguments described as **Latitude** and **ObsGravity** and returning a value described as **FreeAir**

FAmgal(Latitude,ObsGravity m/sec^2) with two arguments described as **Latitude** and **ObsGravity m/sec^2** and returning a value described as **FreeAir**

New specifications for DLL/SO functions

The new type of DLL/SO function returns its values in arguments rather than as the value of the function. The returned value of the function is now a flag showing whether there was a problem in execution. This will increase the versatility of DLL/SO functions. This design will enable functions to have a number of output arguments as well as a number of input arguments.

Here is a brief programmer's summary of the new system, including an example:

With this convention you can prototype all functions in the same way.

The function call

```
function_name (input1,input2, ...)
```

would require

```
int function_name(int no_of_points,double** bufs,int inbufs,int
onbufs)
```

Where

no_of_points is the number of data points in the group represented by the arguments

bufs is an array of pointers each pointing to an array of numbers, one for each input (**inbufs**) and one for each output (Currently **onbufs** must be **1**) .

Example

```
/*
 * Sample program demonstating the use of dynamic library calls
 * from within the Intrepid spreadsheet.
 *
 * The prototype for a function xxx(f1,f2,...,fn) is
 * int xxx(int npts,double** bufs,int inbufs,int onbufs)
 *
 * The install.cfg can have an entry for this module that helps
 * intrepid to find it.
 * If there is no install.cfg entry for this routine it must be
 * put into a shared library called libFA.so.
 * It is assumed that the library will be found on the library
 * path ($LD_LIBRARY_PATH) on UNIX.
 *
 * This module is designed to be called from dbedit as FA(lat,obsgravity).
 * lat is in latitude degrees.
 * obsgravity is in mgals.
 * Does IGSN71 Free Air Anomaly calculation
 *
 */
```

```
#include <stdio.h>
#include <math.h>
int FA_init(int npts,double** bufs,int inbufs,int onbufs) {
  return(0);
}
int FA_final() {
  return(0);
}
int FA(int npts,double** bufs,int inbufs,int onbufs) {
        double* lat = bufs[0];
        double* obsgravity = bufs[1];
        double* FAvalue = bufs[2];
        double PI=3.14159265358979323846;
        double flt64Null = -5.0E+75;    /* This represents a null value */
        int i;
        /* Perform some calculation */
        double a1=978031.85;
        double a2= 0.005278895;
        double a3= 0.000023462;
        double sl,calcgrav;
        for (i=0;i<npts;i++) {
                if (lat[i]!=flt64Null && obsgravity[i]!=flt64Null) {
                        sl = sin(lat[i]*PI/180.0);
                        calcgrav = (a1*(1.0+a2*sl*sl+a3*sl*sl*sl*sl));
                        FAvalue[i] = obsgravity[i]-calcgrav;
                }
                else FAvalue[i] = flt64Null;
        }
        return(0);
```

Notes

- The returned value is now an integer indicating the status of the function (**0** = OK, all other values indicate an error)
- *ERMapper* C routines can be called if they have an entry in the **ERMapper** segment of **install.cfg**
- You can now define the initialiser and finaliser routines as in *ERMapper*. For example:
  - **FA_init** is called at the start of the operation
  - **FA** is called once for each group
  - **FA_final** is called at the end of the operation

For more further assistance with this feature of INTREPID contact our technical support service.

API function list

This section contains a detailed description of the available INTREPID application programming interface (API) functions.

File

int DFAOpenDirectory(dbpath,dir)

Open directory to access data. Directory statistics such as number of files(dir->nfields) are recorded in the INTREPID directory field.

* Parameter dbpath The directory to be opened

* Parameter dir The INTREPID directory field

* Return int The success of the operation 0 success -1 failure

int DFACloseDirectory(dir)

Close directory to access the data

* Parameter dir The INTREPID directory field

* Return int The success of the operation 0 success -1 failure

int DFAOpen(path,io)

Open access to the data

* Parameter path The path to the data

* Parameter io The INTREPID field

* Return int The success of the operation 0 success -1 failure

int DFAClose(io)

Close access to the data

* Parameter io The INTREPID field

* Return int The success of the operation 0 success -1 failure

int DFACreateLike(newfile,likefile,dtype,nbands)

Create a new database file based on another file

* Parameter newfile The path to the file to be created

* Parameter likefile The path to the file to be copied

* Parameter dtype The type of data

* Parameter nbands The number of bands in the data(normally 1)

* Parameter groupby The field of the copied group(normally 1)

* Return int The success of the operation 0 success -1 failure

int DFACreateGroupBy(path,dtype,nbands)

Create a new group(..LINE, .PD and .vec files)

* Parameter path The path to the new file

* Parameter fileType The type of file

* Parameter dtype The type of data

* Parameter nbands The number of bands in the data(normally 1)

* Return int The success of the operation 0 success -1 failure

int DFACreate(path,dtype,nbands)

Create new INTREPID field

* Parameters & Return As above

char DFAPathOfAlias(dir,alias);

Check the pathname is valid

* Parameter dir The INTREPID directory field

* Parameter alias The name of the file

* Return char The pathname if ok or NULL if alias not found

char DFAReadDirectory(dir,fieldnumber);

Read directory

* Parameter dir The INTREPID directory field

* Parameter fieldnumber The field number(ie corresponds to a file)

* Return char The pathname if ok or NULL if fieldnumber is out of range

char DFAGetProjection(io);

Get INTREPID Projection

* Parameter io The INTREPID field

* Return char The Projection from the .vec file for the field. Returns NULL if the .vec file is not found or if there is no projection for the field.

char DFAGetDatum(io);

Get INTREPID Datum

* Parameter io The INTREPID field

* Return char The Datum from the .vec file for the field. Returns NULL if the .vec file is not found or if there is no datum for the field.

int DFAGetNsForGroup(io,group);

Get number of samples for specified group

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Return char Either error==-1 or number of samples for group

int DFASetProjection(io,proj,datum);

Set INTREPID Projection and Datum

* Parameter io The INTREPID field

* Parameter proj The Projection to be set

* Parameter datum The Datum of the projection

* Return int The success of the operation 0 success -1 failure

datatype DFAGetDataType(dir,fieldnumber);

Determine the datatype of the field

* Parameter dir The INTREPID directory field

* Parameter fieldnumber The fieldnumber(ie corresponds to a file)

* Return DataType The datatype of the field or DT_UNKNOWN if an error.

filetype DFAGetFileType(dir,fieldnumber);

Determine the filetype of the field

* Parameter dir The INTREPID directory field

* Parameter fieldnumber The fieldnumber(ie corresponds to a file)

* Return FileType The filetype of the field or FT_UNKNOWN if an error. FileType may be of 4 types: line, image, polygon or point.

Read

int DFAReadR4(io,group,fromrow,torow,buf)

Read real*4 precison data

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Parameter fromrow The start index for part of a group

* Parameter torow The end index for part of a group

* Parameter buf The buffer for io->nb bands

* Return int Either error==-1 or number of samples/rows read

int DFAReadR8(io,group,fromrow,torow,buf)

Read real*8 precison data

* Parameters & Return As above

int DFAReadCH(io,group,fromrow,torow,buf)

Read char type data

* Parameters & Return As above

int DFAReadI2(io,group,fromrow,torow,buf)

Read integer*2 precison data

* Parameters & Return As above

int DFAReadI4(io,group,fromrow,torow,buf)

Read integer*4 precison data

* Parameters & Return As above

int DFAReadBandR4(io,line,band,fromrow,torow,buf)

Read band of real*4 precison data

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Parameter band The band to be read

* Parameter fromrow The start index for part of a group

* Parameter torow The end index for part of a group

* Parameter buf The buffer for io->nb bands

* Return int Either error==-1 or number of samples/rows read

int DFAReadBandR8(io,line,band,fromrow,torow,buf)

Read band of real*8 precison data

* Parameters & Return As above

int DFAReadBandI2(io,line,band,fromrow,torow,buf)

Read band of integer*2 precison data

* Parameters & Return As above

int DFAReadBandI4(io,line,band,fromrow,torow,buf)

Read band of integer*4 precison data

* Parameters & Return As above

Write

int DFAWriteR4(io,group,fromrow,torow,buf)

Write real*4 precison data

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Parameter fromrow The start index for part of a group

* Parameter torow The end index for part of a group

* Parameter buf The buffer for io->nb bands

* Return int Either error==-1 or number of samples/rows read

int DFAWriteR8(io,group,fromrow,torow,buf)

Write real*8 precison data

* Parameters & Return As above

int DFAWriteCH(io,group,fromrow,torow,buf)

Write char type data

* Parameters & Return As above

int DFAWriteBandR4(io,group,band,fromrow,torow,buf)

Write band of real*4 precison data

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Parameter band The band to be read

* Parameter fromrow The start index for part of a group

* Parameter torow The end index for part of a group

* Parameter buf The buffer for io->nb bands

* Return int Either error==-1 or number of samples/rows read

int DFAWriteBandR8(io,group,band,fromrow,torow,buf)

Write band of real*8 precison data

* Parameters & Return As above

int DFAAppendR4(io,group,npts,buf)

Append real*4 precison data

* Parameter io The INTREPID field

* Parameter group The 0 relative implicit group number

* Parameter npts The number of points to be appended

* Parameter buf The buffer for io->nb bands

* Return int Either error==-1 or number of samples/rows appended

int DFAAppendR8(io,group,npts,buf)

Append real*8 precison data

* Parameters & Return As above

int DFAAppendCH(io,group,npts,buf)

Append char type data

* Parameters & Return As above

Error Handling

void DFAPrintError(error);

Print error type to output

* @param error Type of error

void DFAPrintError1(error,message);

Print error type and description to output

* Parameter error Type of error

* Parameter mess1 Description of error